MAC/65



START ADRESSE = $8800

MAC / 65 Beschreibung.

Der MAC/65 gibt es auf Diskette und auf ROM. Beide Versionen sind annähernd identisch. Diese Beschreibung gilt für die Diskversion. Auf der Systemdisk sind noch ein Debbuger (BUG/65) und einige Beispielprogramme. MAC/65 sollte nur mit dem sogenannten "COM-DOS" (OS/A+) der Firma OSS benutzt werden, da MAC/65 auf dieses DOS ausgerichtet ist. MAC/65 läuft auf allen ATARI's mit mindestens 48k RAM. Zum Anfang noch ein paar benutzte Ausdrücke welche bei der Funktionenbeschreibung benutzt werden:

lno - Zeilennummer zwischen 0 und 65535 inclusive. (lno stammt von linenumber)

hxnum - Hexzahl. Adresse oder Wert, integer ohne Vohrzeichen. (hxnum stammt von hex number)

dcnum - Dezimalzahl zwischen 0 und 65535. Dezimalzahlen werden auf zwei Bytes gerundet. So ergibt 3.5 den Wert 4 und 100.1 den Wert 100, dcnum hat kein Vorzeichen. (dcnum stammt von decimal numbers)

exp - Ein assembler Ausdruck. Z.B LDA, .OPT usw. (exp stammt von expression)

string - Alphanummerischer ASCII Ausdruck in zwei Anführungsstrichen eingebettet. Z.B "Hallo".

strvar - Wie string nur ohne Anfürungsstriche. Kann alles sein wei z.B. #$10 oder ein Macro. (strvar stammt von string representation)

filespec oder file - Alphanummerischer ASCII Ausdruck der ein Gerät zur Ein-Ausgabe bestimmt. Z.B. #D:SCREEN.ASM. (filespec stammt von file specific)

Haken zeigen an, daß der nachfolgende Ausdruck nur eingegeben werden muß, wenn er auch benötigt wird.
Z.B. .WORD exp [exp.....]. Pünktchen zeigen an das der Ausdruck sooft wie nötig eingegeben werden soll wie gewünscht wird. Klammern zeigen an das einer der Ausdrücke benutzt werden kann. Z.B (,Q) (,A).
Zum Editor generell: MAC/65 unterscheidet zwischen TEXT MODE und EDIT MODE.

TEXT MODE

Im Einschaltmoment ist der EDIT MODE aktiviert. Geben Sie TEXT ein um in den TEXT MODE zu gelangen. In diesem Mode wird kein Syntaxcheck ausgeführt, was heißt Sie können alles eingeben was Sie wollen MAC/65 nimmt es an. Alle (nachfolgend beschriebenen Editorfunktionen funktionieren im TEXT MODE.

EDITOR MODE

In diesem Mode können Sie Ihr Assemblerprogramm schreiben. MAC/65 überprüft sofort eine eingegebene Zeile oder Kommando auf Richtigkeit, und gibt einen Fehler aus sollte etwas nicht stimmen. Erkennbar ist dieser Mode am Wort EDIT.

EDITOR COMMANDOS

Kommando: **ASM** [#filespec1],[#filespec2],[#filespec3]

Zweck : Assembliert Quelltext, erstellt Listing
und produziert (wenn gewünscht Sourcecode).

filespec1 = Arbeitsspeicher
filespec2 = Bildschirm
filespec3 = Speicher (Achtung: Siehe Note)

Wenn Sie nur ASM eingeben, so gilt die oben genannte Einstellung.

Beispiel: ASM #D2:SOUCRE,#D:LIST,#D2:OBJECT

Hier kommt der Quellcode von D2:SOUCRE, das Listing wird unter D:LIST abgespeichert und der Objektcode wird in D2:SOURCE geschrieben.

Beispiel: ASM #D:SOURCE,,#D:OBJECT

Hier wird der Quellcode von D:SOURCE gelesen, das Listing wird am Bildschirm gezeigt, und der Objektcode wird unter D:OBJECT abgelegt.

Beispiel: ASM ,#P:

Quelltext stammt aus dem Speicher, Objektcode wird in den Speicher geschrieben und das Listing erscheint auf dem Drucker.

Note : Ein Objektcode wird nur produziert, wenn im Assemblerprogramm die Anweisung .OPT OBJ enthalten ist.

Kommando: **BLOAD** #filespec

Zweck : Binärfile laden (OS/A+ Format)

Kommando: **BSAVE** #filespec < hxnum1, hxnum2

Zweck : Savet definierten Speicher als Binärfile ab.
(OS/A+ Format)

Beispiel: BSAVE #D:OBJECT<5000,5100

Der Speicherbreich zwischen $5000 und $5100 wird auf Disk unter dem Namen "OBJECT" abgelegt.

Kommando: **BYE**

Zweck : Sprung zum ATARI Memo Pad (bei 800)
oder Selftest (bei 800XL).

Kommando: **C** hxnum1< (,) (hxnum) [(,)(,hxnum)....]

Zweck : Direkter Speicherzugriff

Beispiel: C 5000<20,00,DB,,5

Bei diesem Beispiel geschiet folgendes:

$5000 bekommt den Wert $20,
$5001 " " " $00,
$5002 " " " $DB,
$5003 bleibt unverändert und
$5004 bekommt den Wert $05 zugeordnet.

Kommando: D hxnum1 [, hxnum2]

Zweck : Zeigt Speicherinhalt in Hexzahlen an.

Wenn hxnum2 nicht angegeben ist, dann werden nur die darauffolgenden acht Werte angezeigt.

Kommando: DEL lno1 [,lno2]

Zweck : Löschen von Zeilennummern

Note : lno1 muß im Speicher stehen.

Kommando: DOS

Zweck : Zurück zum DOS um z.B. die Directory zu sehen.

Note : RUN startet MAC/65 wieder.

Kommando: ENTER #filespec [(,M) (,A)]

Zweck : Läd ASCII oder ATASCII Datei ein.

Note : Parameter "M" bedeutet Merge=verschmelzen die neue Datei wird an die sich im Speicher befindliche angehangen. Alte Zeilen werden durch neue ersetzt.
Parameter "A" ermöglicht ein unnummeriertes Programm "einzuENTERn". ACHTUNG "A" löscht erst den Speicher bevor geladen wird.

Kommando: FIND /string/ [lno1] [,lno2] [,A]

Zweck : Suchen eines bestimmten Ausdrucks im Quelltext.

Beispiel: FIND /LDX/

Es wird das <u>Erste</u> "LDX" gesucht.

Beispiel: FIND /Label/25,80

Es wird das <u>Erste</u> "Label" zwischen den Zeilen 25 bis 80 gesucht.

Note : Wenn die Option "A" gegeben ist, werden alle gefundenen Strings in dem angegebenen Abschnitt angezeigt.

Kommando: LIST [#filespec, [[lno1],[lno2]]

Zweck : Listen von Quelltextzeilen.

Beispiel: LIST #P:

Das gesamte Assemblerprogramm wird auf dem Drucker gelistet.

Beispiel: LIST #D:TEMP,1030,1800

Listet nur die Zeilen 1030 bis 1800 auf Disk zwei unter dem Namen "TEMP".

Kommando: LOAD #filespec [,A]

Zweck : Laden eines Assemblerprogrammes.

Note : Parameter "A" bewirkt das der Speicher nicht gelöscht wird. Alte Zeilen werden nicht überschrieben. D.H. es können Zeilennummern doppelt auftreten.

Kommando: LOMEM hxnum

Zweck : Betimmung der Startadresse des Quelltextes.

Note : Quelltext wird gelöscht.

Kommando: NEW

Zweck : Quelltext und wird gelöscht, EDIT MODE wird gesetzt.

Note : Verläßt TEXT MODE.

Kommando: NUM [[dcnum1] [,dcnum2]]

Zweck : Automatische Zeilennummerierung.

Beispiel: NUM 1000,20

Startzeile ist 2000, es wird in 20ziger Schritten gezählt.

Wenn nur NUM angegeben wird, dann wird die letzte sich im Speicher befindliche Zeilenummer plus zehn angezeigt. Wird nur dcnum1 angegeben, so wird von der letzten Zeilennummer + dcnum1 an gezählt, in dcnum1 Schritten.

Note : BREAK bricht Autonummerierung ab.

Kommando: PRINT [#filespec,] [lno1], [lno2]]

Zweck : Siehe LIST Kommando. Unterschied: Es werden keine Zeilennummern gelistet.

Kommando: REN [dcnum1] [,dcnum2]

Zweck : Umnummerierung von Quelltextzeilen.

Die erste Zeilenummer ist dcnumm1, incrementiert wird mit dcnum2. Werden keine Werte angegeben so wird dcnum1 und dcnum2 auf 10 angenommen.

Note : Die Zeilenummer dcnum1 muß im Speicher stehen!

Kommando:
REP /old string/new string/ [[lno1] [lno2]] [(,A) (,Q)]

Zweck : Ersetzen eines Ausdrucks durch einen Anderen.

Parameter "A" bedeutet das alle gefundenen "old strings" in dem angegebenen Raum ersetzt werden. Parameter "Q" bedeutet das vor jeder Substitution gefragt wird. (Y+Taste RETURN bedeutet ersetzen, nur Taste RETURN heißt nicht ersetzen). Werden keine Parameter angegeben, so wird nur der erste Ausdruck ersetzt.

Beispiel: REP /LDY/LDA/200,250,Q

Dieses Beispiel durchsucht die Zeilen 200 bis 250 nach dem Ausdruck "LDY", und fragt (wenn gefunden), ob ersetzt werden soll.

Note : Taste BREAK bricht die Funktion ab.

Kommando: **SAVE** #filespec

Zweck : Speichert Assemblerprogramm auf das angegebene Gerät ab.

Kommando: **SIZE**

Zweck : Anzeige von drei Werten deren Bedeutung ist:

Wert1 = Niedrigste benutzbare Speicheradresse (kann mit Kommando LOMEM gesetzt werden)
Wert2 = Höhste vom User benutzte Adresse
Wert3 = Höhste benutzbare Adresse

Note : Alle Werte sind Hexadezimal.

Kommando: **TEXT**

Zweck : Erlaubt es dem Benutzer alles einzugeben was er will, es erfolgt kein Syntaxcheck.

Note : Kommando NEW bricht TEXT MODE ab. Alle Daten werden gelöscht.

Kommando: **?**

Zweck : Zahlen Umrechnung.

Beispiel: ? $1200 ergibt =4608
? 8190 ergibt =$1FFE

Als letztes Kommando, sei der

MAC65 [file1] [file2] [file3] [[-A] [-D]]

Befehl genannt. Er hat den gleichen Zweck wie das ASM Kommando, mit dem Unterschied das dieser Befehl direkt aus dem DOS eingegeben werden muß.

Option "-A" bedeutet das der Quelltext in ASCII gehalten ist, Option "-D" heißt das eine Disk to Disk Assemblierung gewünscht ist. Mit dieser Option können Quelltexte die län-

ger sind als der Hauptspeicher fassen kann bearbeitet werden
MAC/65 bietet auch eine Menge Operatoren an, als da wären:

Operatoren: + - * /

Plus - Minus - Mal - Geteilt, normale Operatoren wie gewohnt (16 Bit).

Beispiel: $FF00+4096 ergibt $0F00, es wird kein Überlauf angezeigt.

Operatoren: & ! ^

AND - OR - EXOR (16 BIT) Bitweise.

```
Beispiel: $FF00 & $00FF ergibt $0000
          $03 ! $0A     ergibt $000B
          $003F ^ $011F ergibt $0120
```

Operatoren: = > < <> >= <=

Gleich - Größer - Kleiner - Ungleich
Größer Gleich - Kleiner Gleich

```
Beispiel: 3<5  ergibt 1
          5<5  ergibt 0
          5<=5 ergibt 1
```

Operatoren: .OR .AND .NOT

Logische Operationen.

```
Beispiel: 3 .OR 1  ergibt 1
          3 .AND 2 ergibt 1
          6 .AND 0 ergibt 0
            .NOT 7 ergibt 0
```

Operator : -

Negatives Vorzeichen.

Beispiel: -2 ist gleich mit $FFFE (0-2)

Operatoren: < >

Zerlegt 16 Bit Wert in High und Low Teil.

```
Beispiel: FLEEP=$3456
          LDA #<FLEEP  (entspricht LDA #$56)
          LDA #>FLEEP  (entspricht LDA #$34)
```

Operator : .DEF

Dieser Befehl testet ob der nachfolgende Label bis dahin schon definiert wurde.

```
Beispiel:     .IF .DEF ZILK
              .BYTE "ein paar Bytes"
              .ENDIF
             ZILK = $3000
```

Als Wert übergibt DEF die Zahl 0 (nein), denn der Label "ZILK" wurde bis dahin noch nicht deklariert. ACHTUNG: Da

ZILK danach aber bestimmt wird, sieht MAC65 im zweiten Assemblierungsdurchgang das es doch den Label "ZILK" gibt, MAC65 meldet dann "OUT OF PHASE". Bei .DEF gilt, entweder der Label wird vor dem .DEF deklariert oder garnicht.

Note : In diesem Beispiel wird der Befehl .BYTE "ein paar Bytes" nicht assembliert. (Da .DEF ja den Wert Null ergibt).

Operator : .REF

Beispiel:

```
        .IF .REF PRINTMSG
      PRINTMSG
        (Code
           der PRINTMSG
             Routine)
        .ENDIF
```

Für den Operator REF gilt das Gleiche wie für DEF, mit dem unterschied, das REF den Wert eins ergibt wenn der nachfolgende Label auch benutzt wurde, z.B durch:
LDA #<PRINTMSG,
JMP PRINTMSG,
JSR PRINTMSG usw.

Dieser Befehl ist ein effizienter Weg mehrere Routinen bereit zu haben, und nur die benötigten zu Assemblieren.

Operatoren: [ ]

Beispiel:

```
LDA GEORGE+5*3        ;ist erlaubt, es wird 3*5
                       gerechnet und zum Label
                       GEORGE addiert.

LDA [GEORGE+5]*3      ;erst wird die 5 addiert,
                       und dann der Wert mit 3
                       multipliziert.

LDA ([GEORGE+5]*3),Y;ist ok.
```

Operatoren können beliebig zu einem Assemblerausdruck zusammengestellt werden, MAC65 bestimmt nach den oben vorgestellten Kriterien eine 16 Bit große Zahl (ohne Vorzeichen).
Zu den Steuerbefehlen. Solche Befehle werden auch Pseudoassemblerbefehle genannt, welche nicht mit 6502 Befehlen zu verwechseln sind.

Befehl : *=

Zweck : Bestimmung wo der folgende Quellcode "hinassembliert" werden soll.

Beispiel: 50 *= $1234

Setzt Zeiger auf $1234
Diese Anweisung wird auch dazu verwendet, um mehreren Bytes Platz zu schaffen.

Beispiel: 70 LOC *=*+5

Reserviert fünf Bytes für den Label "LOC".

Befehl : .DBYTE

Zweck : Weist Speicher eine 16Bit Zahl zu.

Beispiel: 10 .DBYTE $1234,1,-1

Es werden folgende Bytes erzeugt: 12 34 00 01 FF FF

Note : Siehe auch Befehl ".WORD"

Befehl : .ELSE

Note : Siehe ".IF" Befehl zur Erläutrung.

Befehl : .END

Zweck : Programmende Anzeige.

Note : Es wird bei .END nur die Assemblierung beendet wenn aus dem Speicher Assembliert wird. (Ist nötig um den Befehl .INCLUDE richtig einsetzen zu können)

Befehl : .ENDIF

Zweck : Beendet eine definierten Programmblock.

Note : Siehe ".IF" Befehl zur Erläutrung.

Befehl : .ERROR string

Zweck : Erzeugung eines Pseudoerrors, bei dem der nachfolgende string ausgegeben wird.

Beispiel: 100 .ERROR "FEHLENDE PARAMETER"

Befehl : .IF exp
[.ELSE]
.ENDIF

Zweck : Entscheidung ob der nachfolgende Quelltext assembliert wird, als Kriterium gilt der Wert des nachfolgenden Operand. (1=ja=war bzw. truth).

Wenn eine ".IF" Anweisung gefunden wird,wird getestet ob exp null oder eins ist. Ist exp gleich null so wird der Programmteil zwischen ".ELSE" und ".ENDIF" assembliert. Gibt es keine ".ELSE" Anweisung, so wird nichts zwischen ".IF" und ".ENDIF" assembliert. Hat exp den Wert eins, so wird das Programmteil zwischen ".IF" und ".ELSE" übersetzt. Wurde kein ".ELSE" Befehl verwendet, so wird alles zwischen ".IF" und ".ENDIF" assembliert.

Beispiel:
```
10 .IF 1        ;exp ist eins
20 LDA #'?
30 JSR CHAROUT
40 .ENDIF
```

Da der IF-Test immer positiv ausfällt (exp ist 1) werden die Zeilen 20 und 30 immer assembliert. Siehe auch ".REF" und ".DEF" Operanten.

Note   : Maximale Verschatelungstiefe sind 14 offene ".IF" Anweisungen. Fehlende ".ENDIF" Befehle werden nich angezeigt.

Befehl   : .INCLUDE #filespec

Zweck   : Einbinden eines Assemblerprogrammes in ein Anderes.

Beispiel:
```
10 ; Programm 1
20 LDA #10
30 STA $D100
40 .INCLUDE#D:PROGRAM2.ASM
50 JSR SEND
60 .END
```

Es werden die Zeilen 10 bis 30 bearbeitet, dann wird von Diskette das Programm "PROGRAM".ASM" (ran)assembliert danach werden die letzten Zeilen abgehandelt.

Note   : Das "includtetes" Programm wird nur solange assembliert bis ein ".END" Befehl gefunden wird, oder es zu Ende ist. Ein "includetes" Programm darf seinerseits kein anderes Programm "includen"!

Befehl   : .LOCAL

Zweck   : Erklärt neue locale Zone, und schließt Alte ab.

Locale Variablen gelten nur für einen durch ".LOCAL" definierten Bereich. Somit können die gleichen Labelnamen mehrfach verwendet werden. Locale Labels haben als erstes Zeichen immer ein "?".

Beispiel:
```
10  *= $4000
11  LDX #3
12 ?LOOP
13  LDA VON,X
14  STA NACH,X
15  DEX
16  BPL ?LOOP
17 ;
18 .LOCAL        ; neue Local Region
19 ;
20 ?LOOP=6
21 ;
22  LDY #?LOOP ; entspricht LDY #6
23  (usw.)
```

Note   : Locale Labels werden in der symbol table nicht angezeigt.

Befehl   : .OPT option
(oder)
.OPT NO option

Zweck   : Steuerung von Assemblerfunktionen bei der Assemblierung.

Erlaubte optionen sind:

LIST ERR EJECT OBJ MLIST CLIST NUM WAIT

LIST : Assemblerlisting wird angezeigt.
NO LIST : Assemblerlisting wird nicht angezeigt.
ERR : Auftretende Fehler werden angezeigt.
NO ERR : Auftretende Fehler werden nicht angezeigt..
EJECT : Es wird über jede Seite eine Seitenzahl angezeigt, und falls gewünscht ein Titel.
NO EJECT: Es wird über jede Seite keine Seitenzahl und kein Titel gezeigt. (NO EJECT hat keinen Einfluß auf den ".PAGE" Befehl)
OBJ : Es wird ein Objectcode in den Speicher bzw. zum angegebenen Gerät gesendet.
NO OBJ : Es wird kein Objectcode generiert. Gut für Versuchsassemblierungen.
NUM : Das ausgegebene Assemblerlisting wird von 100 an in Einerschritten durchnummeriert. (Nur die Ausgabe nicht das Programm).
NO NUM : Es werden die verwendeten Zeilennummern angezeigt.
MLIST : Listet alle Macrozeilen.
NO MLIST: Listet nur die assemblierten Macrozeilen.
CLIST : Listet alle Zeilen.
NO CLIST: Listet nur die assemblierten Zeilen.
WAIT : Wartet bevor Seitennummer und Titel angezeigt werden.(Nur sinnvoll wenn ein Drucker mit Einzelblatt verwendet wird). START setzt Assemblierung fort.
NO WAIT : Alles wird in einem Stück (durch)assembliert.

Werden keine Optionen angegeben, so gilt:

LIST	es wird ein Listing ausgegeben
ERR	Errors werden angezeigt
EJECT	Seitennummer und Titel werden angezeigt
NO NUM	verwendete Zeilennummern werden angezeigt
MLIST	alle Macrozeilen werden gelistet
CLIST	alle bedingten Zeilen werden gezeigt
NO WAIT	es wird nicht gewartet
NO OBJ	siehe Note!

Note : OBJ ist zweifach zu sehen:
Wird in den Speicher assembliert gilt NO OBJ.
Wird zu einem Gerät assembliert, so gilt OBJ.

Befehl : .PAGE [string]

Zweck : Überschrift.

Der nachfolgende String wird unter die Seitenzahl und dem Titel gedruckt, außerdem wird die nächste Seite angefangen.

Befehl : .SBYTE

Zweck : Produziert Bildschirmkompatible Bytes.

Note : Siehe Befehl ".BYTE" zur Erklärung.

Befehl : .SET dcnum1,dcnum2

Zweck : Kontrolle verschiedener Assemblerfunktionen

dcnum1 dcnum2 Funktion:

0	(4) 1-4	Setzt die Anzahl der Objectbytes die bei den Befehlen .BYTE und .SBYTE nebeneinander im Object-feld des Assemblerlistings dargestellt werden.
1	(0) 0-31	Setzt den linken Rand des Assemblerlistings (Anzahl der Space die vor jeder Zeile stehen).

Note : Die Werte in den Klammern stellen die Standarteinstellung dar.

Befehl : .TITLE string

Zweck : Bestimmung der Überschrift die neben der Seiten-angabe stehen soll.

Befehl : .WORD

Zweck : Weist Speicher eine 16Bit Zahl zu.

Im Unterschied zum ".DBYTE" Befehl wird hier die Zahl in 6502 LSB/MSB Form genneriert.

Beispiel: .WORD $1234,1,-1

Es werden folgende Bytes genneriert: 34 12 01 00 FF FF

Zu den Macros. Ein Macro ist eine Serie von Assemblerbe-fehlszeilen welche einen Namen zugeordnet bekommen. Wenn MAC65 dann diesen Namen im Befehlsfeld findet, werden dort diese Zeilen assembliert. Macros können auch Parameter über-geben werden, welche dann bei assemblieren eingesetzt werden.

Befehl : .ENDM

Zweck : Ende eines definierten Macros

Befehl : .MACRO macroname

Zweck : Start eines Macros.

Alle Zeilen zwischen .MACRO und .ENDM werden abgespeichert im Macrospeicher. Ein Macro kann Seinerseits nich auch ein Macro enthalten.

Beispiel:
```
10    .MACRO PUSHXY
20    TXA
30    PHA
40    TYA
50    PHA
60    .ENDM
```

Sollte jetzt der Befehl "PUSHXY" eingegeben werden, so wer-den die Zeilen 20 bis 50 assembliert.
Alle benutzten Labels in einem Macro, sind locale Labels,

und gelten nur für das Macro. Alle Labelwertzuordnungen durch den Befehl "=", werden wie der Befehl ".=" behandelt.

Macroparameter: Es besteht die Möglichkeit Macro's Parameter zu übergeben, welche dann bei der Assemblierung eingesetzt werden. (Maximal 63 Parameter pro Macro sind erlaubt). Es wird zwischen Zahlenparameter und Zeichenparameter unterschieden. Zahlenparameter werden durch "%" gekennzeichnet, und Zeichenparameter durch "%$".

Beispiel :

```
10  .MACRO 1
20  LDA #>%1  ;Es wird das High-Byte von Para-
                 meter 1 eingesetzt.
30  CMP (%2,X) ;Parameter 2 wird eingesetzt.
40 SYMBOL.=SYMBOL+1
50 LDX #%(SYMBOL) ;Der Parameter mit der Num-
                   mer des Labels "SYMBOL"
                   wird eingesetzt.
60 .BYTE %$1,%2,0 ; Es werden die Stringpara-
                     meter 1 und 2 eingesetzt,
                     und danach der Wert null.
```

Parameter %0 hat eine besondere Bedeutung, er gibt an wieviel Parameter übergeben werden. (Z.B. wird ein Macro assembliert welches 3 Parameter braucht, es werden aber nur 2 übergeben, mit Parameter %0 kann dieser Fehler leicht erkannt und abgefangen werden. Mit Hilfe diese Parameter können auch Makros effizienter programmiert werden.

Beispiel:

```
10  .MACRO PRINT
20  ; DIESES MACRO GIBT EINEN STRING
30  ; AUF DEM BILDSCHIRM AUS.
40  .IF %0 = 1 ; WURDE EIN STRING ÜBERGEBEN?
50  JMP LBL1   ; JA,SPRINGE UND
60  STRING .BYTE %1,EOL; SPEICHERE STRING AB
70  ;
80  LBL1
90  LDX #>STRING ;ADRESSE DES STRINGS IN X&Y
100 LDY #<STRING
110 JSR STRINGOUT ;UND AUSGEBEN
120 .ELSE         ;ANSONSTEN WURDE KEIN STRING
130 LDA #EOL      ;ÜBERGEBEN, ALSO NUR EIN EOL
140 JSR CHAROUT   ;AUSGEBEN
150 ;
160 .ENDIF
170 .ENDM
```

Um das Macro "PRINT" zu nutzen ein Beispiel:

```
1000 PRINT "HALLO, MEIN MACRO FUNKTIONIERT"
1010 PRINT
```

Zu guter letzt noch die Fehlermeldungen, und dann ist alles geschafft:

1 - MEMORY FULL : Speicher ist voll

2 - INVALID DELETE : Die erste Zeilennummer des "DEL" Befehls muß im Speicher stehen bzw. die zweite Zeilennummer muß größer sein als die Erste.

3 - BRANCH ERROR : Ein verwendeter Branch-Befehl kann die angegebene Distanz nicht überspringen.

4 - NOT Z-PAGE / IMMADIATE MODE : Ein Zeropage Befehl wurde auf eine Adresse außerhalb des Zeropage angewandt.

5 - UNDEFINED : Ein Undifiniertes Label wurde benutzt.

6 - EXPRESSION TO KOMPLEX : Es wurden zu viele Ausdrücke benutzt (z.B., = .MACRO, Labels). Defenieren Sie ihre Labels temporär mit ".=".

7 - DUPLICATE LABEL : Ein Label wurde mehrfach mit "=" Werte zugeordenet.

8 - BUFFER OVERFLOW : Syntaxbuffer ist voll. Kürzen Sie ihre Programmzeilenlänge.

9 - CONDITIONAL NESTING : Es fehlen zu .IF Anweisungen .ENDIF Befehle.

10 - VALUE > 255 : Eine Rechnung hat einen Wert >255 ergeben, und ein Wert <=255 wird gebraucht.

11 - CONDITIONAL STACK: Zu viele .IF Anweisungen wurden verschatelt programmiert.

12 - NESTED MACRO DEFINITION : Ein Macro enthält ein weiteren MACRO Befehl.

13 - OUT OF PHASE : Ein Label welches im ersten Phase nicht existierte, ist jetzt im zweiten Phase aufgetaucht. (Siehe auch ".REF" Befehl.

14 - *= EXPRESSION UNDEFINED : Der Programmzähler wurde rückwärts gesetzt, so wird z.B. ein eben assemblierter Objectcode überschrieben.

15 - SYNTAX OVERFLOW : Die Zeile ist zu lang als das MAC65 einen Syntaxcheck durchführen kann.

16 - DUPLICATE MACRO NAME : Es wurden zwei Macros mit dem gleichen Namen definiert. Nur

der Erste zählt.

17 - LINE # > 65535 : MAC65 akzeptiert nur Zeilennummern <= 65535.

18 - MISSING .ENDM : Es wurde ein EOL gefunden, bevor der ".ENDM" Befehl gefunden wurde. Ein Macro darf sich nicht über mehrere Files erstrecken.

19 - NO ORIGIN : Es wurde kein ".*=" Befehl verwendet. Dieser Fehler tritt nur auf wenn ein Objectcode erstellt wird.

20 - NUM/REN OVERFLOW : Bei einer der beiden Befehle wurde eine Zeilennummer >65535 erzeugt.

21 - NESTED .INCLUDE : Ein includetes File kann seinerseits kein anderes File includen.

22 - LIST OVERFLOW : Der list output Buffer hat mehr als 255 Zeichen empfangen. Nehmen Sie kleinere Werte beim ".TAB" Befehl.

23 - NOT SAVED FILE : Ein zu bearbeitendes File wurde nicht gefunden.

24 - LOAD FILE TO BIG : Das File paßt nicht in den Speicher.

25 - NOT BINARY SAVED : Ein File welches nicht binär ist wurde versucht mit ".BLOAD" zu laden.

30 - UNDEFINED MACRO : Ein Macro was nicht existiert wurde versucht aufzurufen.

31 - MACRO NESTING : Die maximale Verschachtelung eines Macros von 14 Ebenen wurde überschritten.

32 - BAD PARAMETER : Einem Macro wurden Parameter übergeben, obwohl es keine braucht, oder es wurden mehr als 63 Parameter übergeben.

128 - 255 (Normale System Fehlermeldungen) : Zur Erkläng siehe ATARI-Handbuch.

Ach du Schande, ich vergaß einen Befehl zu beschreiben, hier ist er:

Befehl : .TAB dcnum1, dcnum2, dcnum3

Zweck : setzen von Tabulatoren zur Lesbarkeit.

dcnum1 setzt die Startspalte des Instruktionsfeld.

dcnum2 setzt die Startspalte des Operantenfeld.

dcnum3 setzt die Startspalte des Bemerkungsfeld.

```
Beispiel :100   .TAB 16,32,50

          ......

          1000  .TAB 8,12,20   ;setzt standard Werte.
```

Zur näheren Erklärung eine Typische Assemblerzeile:

```
1FB7 8503    1390          STA          CASINIT+1     ;REMARK

                     !Instrucktionen!  !Parameter!  !Bemerk-!
                           Feld           Feld        kungs
                                                      Feld
```